16AE-0046

Automatic functionality assignment to AUTOSAR multicore distributed architectures

Florin Maticu, Paul Pop
Technical University of Denmark, Denmark

Axbrink Christian, Islam Mafijul
Volvo Group Trucks Technology, Sweden

Abstract

The automotive electronic architectures have moved from federated architectures, where one function is implemented in one ECU (Electronic Control Unit), to distributed architectures, where several functions may share resources on an ECU. In addition, multicore ECUs are being adopted because of better performance, cost, size, fault-tolerance and power consumption. In this paper we present an approach for the automatic software functionality assignment to multicore distributed architectures. We consider that the systems use the AUTomotive Open System ARchitecture (AUTOSAR). The functionality is modeled as a set of software components composed of subtasks, called runnables, in AUTOSAR terminology. We have proposed a Simulated Annealing metaheuristic optimization that decides: the (i) mapping of software components to multicore ECUs, (ii) the assignment of runnables to the ECU cores, (iii) the clustering of runnables into tasks and (iv) the mapping of tasks to "OS-Applications" (used to isolate mixed safety-criticality functions). We are interested to determine an implementation such that (1) the mapping constraints are satisfied, (2) the runnables are schedulable and (3) they are spatially and temporally isolated if they have different safety-criticality levels, (4) the overall communication bandwidth is minimized and (5) the utilization of the cores and ECUs is balanced. The proposed approach was evaluated on three realistic case studies.

Introduction

Many embedded applications, following physical, modularity or safety constraints, are implemented using distributed architectures, composed of several different types of hardware components (called Electronic Control Units, or ECUs), interconnected in a network. The application software running on such distributed architectures is composed of several functions. The way the functions have been distributed on the architecture has evolved over time. Initially, in automotive and aerospace applications, for example, each function was running on a dedicated hardware node, allowing the system integrators to purchase nodes implementing required functions from different vendors, and to integrate them together into their system (this approach is also called a "federated architecture"). However, the number of such nodes in the architecture has exploded, reaching over one hundred in an airplane or a high-end car, leading to increased wiring, in-creased costs, size, weight and power consumption.

These trends have created a huge pressure to reduce the number of nodes, use the resources available more efficiently, and thus reduce costs. This is achieved through the integration of several functions in one node (also called an "integrated architecture"), where the nodes are part of a distributed architecture. The vehicle industry has addressed the challenge of integrating more functions by introducing AUTOSAR (AUTomotive Open System ARchitecture) [4], a standardized model of development that makes it possible for software developers to create reusable software components that are hardware independent. In addition, the nodes themselves can be integrated into a single chip, as is the case with the trend towards using multicore architectures, where several processing cores can be integrated onto a single chip, decreasing the costs, power consumption, size, and increasing the performance through parallelization [15]. The AUTOSAR framework has a standardized layered software architecture made of three parts fig. 1:

- An application software layer.
- A middle layer called Runtime Environment (RTE).
- The Basic Software layer (BSW).

The application layer is composed of the *software components* that provide the functionality required on the ECU. The RTE layer defines a standardized application program interface (API) that allows an application to call a service from the Basic Software Layer. Furthermore, the communication between *software components* is also performed via the RTE layer. The Basic Software Layer consists of the Operating System and modules

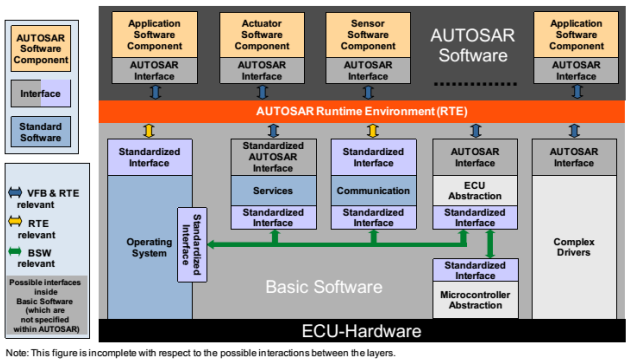


Figure 1: AUTOSAR layers, source:[3]

that provide services like communication over a network, I/O, memory access. It provides an abstract layer to the *software components* that hides the ECU-hardware details. The *software components* logically interact through a Virtual Function Bus (VFB). The Runtime Environment (RTE) can be seen as the implementation of the (VFB) providing the API necessary for them to exchange data signals and access OS services.

Safety is a property of a system that will not endanger human life or the environment. ISO 26262 [13] defines *Automotive Safety-Integrity Levels* (ASILs), which are assigned to safety-related functions to capture the required level of risk reduction, and may dictate the development processes that have to be followed. There are four ASIL levels, ranging from ASIL D (most critical) to ASIL A (least critical); a QM (Quality Management) level means that the function is non-critical. The same trends are driving the integration of several levels of safety-criticality, together with non safety-critical functions. The "Research Agenda for Mixed-Criticality Systems" [5] defines a mixed-criticality system as "an integrated suite of hardware, operating system and middleware services and application software that supports the execution of safety-critical, mission-critical, and non-critical software within a single, secure computing platform". In a mixed-criticality system, the safety functions of different criticality levels are typically *separated* (or, *isolated*); without separation, for example, a lower-criticality task could corrupt the memory of a higher-criticality task.

In this paper, we are interested in the mapping of mixed-criticality applications to distributed heterogeneous multicore architectures. The architectures are composed of ECUs interconnected using the Controller Area Network (CAN) [12]. We have shown in earlier work how other protocols, such as FlexRay [20] or TTEthernet [25], can be taken into account. An AUTOSAR application is composed of a several *software components* interacting via *ports* and consisting of *runnables*. *Software components* have to be mapped to ECUs and *runnables* are mapped to cores. *Runnables* have to be grouped into *Tasks* which then are grouped into *OS-Applications*.

The *software components* provide the functionality required on the ECU. A *port* has a associated *port-interface* that is the *"contract"* between one *software component* that provides the interface (P-ports) and the one that requires an interface (R-Ports). The behavior of a *software component* is constructed using the entities called *runnables*. They are software functions that implement the algorithms (behavior) of a *software component*. Each *runnable* has access to the port interfaces and can read/write data signals from/to other *software components*.In AUTOSAR, each *runnable* execution must be triggered by an event. A *runnable* can start its execution when new data is available to it's associated *software component* port (data receive event) or it can be triggered by a timer (timing event).

The unit of execution inside AUTOSAR Operation System is called an *Task*. Each *Task* has assigned a priority and it can always be preempted by another *Task* with a higher priority value. Each *runnable* from any *software component* needs to be mapped to an *Task*. Multiple *runnables* can be assigned to the same *Task*. According to [22], the simplest solution is to map each *runnable* into its own *Task* but this is not feasible because the number of tasks can be limited in many systems and is not efficient (the core utilization overhead needs to be taken into account when the Operating System switches between tasks).

An *Os-Application* is an AUTOSAR entity that groups together a collection of Os-objects defined as *Tasks*, Interrupt Service Routines, alarms, events, counters, etc. An *Os-Application* can be trusted, which means that each object that is part of it has unrestricted access to the RTE API and hardware resources. Alternatively, each object of an untrusted *Os-Application* has limited access to the RTE API and hardware resources and it runs in non-privileged mode. Each *Os-Application* has its own memory partition, separate stack, data and code. AUTOSAR assures that a code executed in the context of an *Os-Application* can not corrupt the memory area of another *Os-Application*.

Given an application model and an architecture model, in this paper we are interested to solve the mapping problem, which determines: (i) the mapping of software components to multicore ECUs, (ii) the assignment of runnables to the ECU cores, (iii) the clustering of runnables into Tasks and (iv) the mapping of Tasks to OS-Applications. We are interested to determine a mapping such that (1) the mapping constraints are satisfied, (2) the runnables are schedulable and (3) they are spatially and temporally isolated if they have different safety-criticality levels, (4) the overall communication bandwidth is minimized and (5) the utilization of the cores and ECUs is balanced.

To solve this problem, we have proposed a Simulated Annealing-based metaheuristic optimization. The proposed approach has been evaluated on three vehicle case studies, one of them composed of a large set of runnables from a set of applications from Volvo Trucks.

Related work

There is a large amount of research on hard real-time systems [14], including task mapping to heterogeneous architectures [6]. Researchers also addressed the problem of mixed-criticality systems. A recent review of the research in the area of mixed-criticality systems was written by Burns and Davis [7].

The assignment of functions to the distributed vehicle architecture (mapping) is typically done manually, based on the expe-

rience of the systems integrator. However, such a manual mapping is no longer feasible due to the introduction of multicores, which increases the complexity of the decisions and their impact, the use of AUTOSAR and the required compliance to ISO 26262. Without an automatic mapping approach, it is very challenging to utilize multicore-based ECUs in automotive systems, as experienced by Volvo Group, one of the leading manufacturers of commercial vehicles.

The authors of [21] propose an ILP (Integer Linear Programming) approach for mapping AUTOSAR functions on a multicore ECU to minimize the inter-core communication and balance the core load. They consider a single ECU, and the ILP formulation is used only on small examples, since it is unfeasible for larger systems.

An approach for mapping AUTOSAR functionalities on multicore ECU that takes into consideration timing and precedence constraints is proposed in [9]. This work considers also a single ECU. The authors are using a heuristic method called "Systematic Memory based Simulated Annealing" (SMSA) and argue that it provides better results than the classic Simulated Annealing.

Another approach for mapping AUTOSAR functions into a single multicore ECU is presented in [18]. The goal of that work was to balance the core utilization and the strategy used was to cluster functions that are exchanging data signals and iteratively assign them to the least loaded core. For the automotive applications, where most of the functions are exchanging signals, applying this strategy will result in most of them being clustered together and assigned to one core.

In [26], the authors proposed a Mixed Integer Linear Programming and a Genetic Algorithm for mapping AUTOSAR functionalities on a architecture composed of several single core ECUs. The goal is to optimize end-to-end worst-case response times and the memory consumption.

Compared to related work, our approach carefully takes into consideration the details of the scheduling, partitioning and communication models of AUTOSAR. In our case, the schedulability test for tasks takes into account the details of the configuration, which, for example, introduces different communication overheads depending on how runnables are mapped, i.e., same or different task, same or different core or ECU. Although the impact of AUTOSAR on timing has been investigated before [19], the authors do not address the mapping problem. Also, none of the previous work has addressed the issue of mixed-criticality functions.

Architecture Model

This section presents the system architecture model. The next two sections introduce the application model and the AUTOSAR software architecture model.

The hardware architecture consists of heterogeneous multicore and single-core ECUs ECU_i that are connected through a shared network bus (CAN in our case), which has a given *bandwidth*. We denote with N the total number of cores in the system.

$$Architecture = \langle \{ECU_i\}, Bus \rangle \tag{1}$$

Each ECU ECU_i has a set of interconnected cores $\{Core_j\}$ and an associated ASIL level $ASIL_i$.

$$ECU_i = \langle \{Core_j\}, ASIL_i, NoC_i \rangle \tag{2}$$

If an ECU_i has multiple cores, we assume that they are connected by a network-on-chip, NoC_i. $NoC_i(Cores_i, Links_i)$ is a graph, where the vertices $Cores_i$ are the cores on the ECU, and the edges $Links_i$ are the communication links. For each link $Link_{a,b} \in Links_i$ connecting $Core_a$ to $Core_b$, we know the $bandwidth_{a,b}$.

An example of an hardware architecture is presented in fig. 2. There are two ECUs, one with a single core processor and one with a multicore processor connected through a network bus. ECU_1 models a single-core architecture that has a CPU, memory, storage and peripherals. ECU_2 models Freescale's MPC5777M MCU[1], which has two computational cores ($Core_2$ and $Core_3$) and one input/output core (I/O core, $Core_4$) used for handling hardware peripherals. This ECU is ISO 26262 compliant up to ASIL level D.

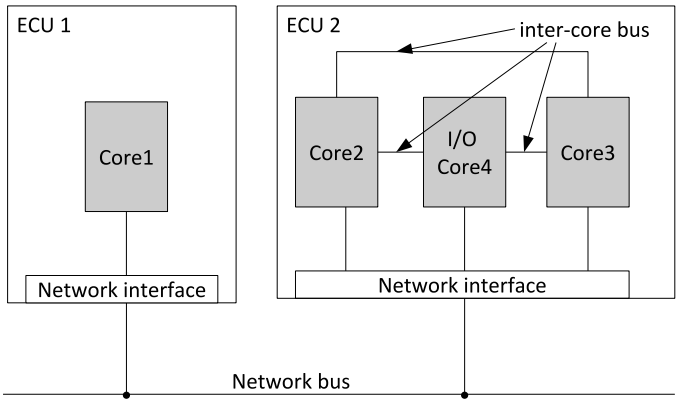


Figure 2: Hardware architecture model

Application Model

We model an application using the modeling concepts of AUTOSAR. Thus, an application is composed of a set of software components, i.e.,:

$$Application = \{Software\ Component_i\}. \tag{3}$$

Each software component contains a set of runnables, and each software component is assigned an ASIL level according to ISO 26262.

$$Software\ Component_i = \langle \{Runnable_j\}, ASIL\ level_i \rangle, \tag{4}$$

A runnable $Runnable_i$ is captured by the n-tuple:

$$Runnable_i = \Big\langle \{WCET_i^j\}, T_i, D_i, ASIL_i, \{\langle Runnable_k, signal_{i,k} \rangle\} \Big\rangle, \tag{5}$$

[1] http://cache.freescale.com/files/32bit/doc/fact_sheet/MPC5777MFS.pdf

where $\{WCET_i^j\}$ is the set of *worst-case execution times* of runnable $Runnable_i$ when executing on the cores $Core_j$ where it is considered for mapping, T_i is the period of a the runnable and D_i is the deadline of the runnable. These elements form the timing model of a runnable. Further, $ASIL_i$ is the ASIL of the runnable, inherited from the parent software component. The data dependencies between runnables are captured by the set $\{\langle Runnable_k, signal_{i,k}\rangle\}$, where each $Runnable_k$ receives the $signal_{i,k}$ from $Runnable_i$. We also know the size $size_{i,k}$ of the signals exchanged.

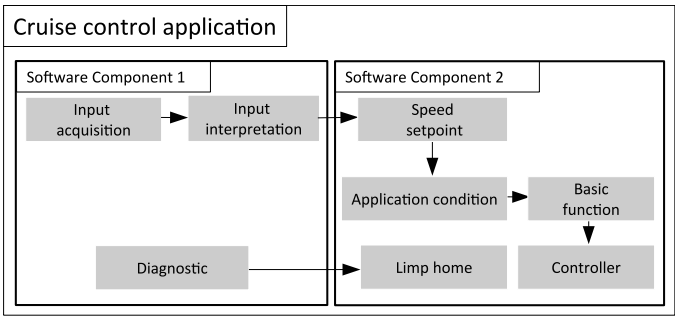


Figure 3: Example application model: cruise control

Table 1: Example application model: runnables

$Runnable_i$	$WCET_i$ (ms)	T_i (ms)	D_i (ms)	$ASIL_i$
Input acquisition	0.5	10	10	A
Input interpretation	1	10	10	A
Diagnostic	1.5	10	10	A
Speed Setpoint	1	10	10	QM
Limp home	1.5	10	10	QM
Basic function	2.5	10	10	QM
Controller	3	10	10	QM

Table 2: Example application model: signals

Sender runnable	Receiver runnable	Size (bytes)
Input acquisition	Input interpretation	2
Input interpretation	Speed setpoint	4
Diagnostic	Limp home	8
Application condition	Basic function	4
Speed setpoint	Application condition	2
Basic function	Controller	8

An example of an application model is presented in fig. 3 and tables 1 and 2. It has been adapted from the automotive use case described in [2]. The application implements the logic for a cruise control system in the vehicle and is composed of two software components. The first software component, "Data handling", contains three runnables responsible for the acquisition of data and diagnostics. The second software component, "Cruise handling", has five runnables controlling the vehicle speed in cruise mode. The arrows between runnables represent the exchanged data signals.

Software Architecture Model

We assume that an AUTOSAR software framework is running on each ECU. The schedulable entity in the AUTOSAR OS is a *Task*. A Task is composed of several runnables, and it is modeled by the following n-tuple:

$$Task_i = \langle WCET_i, T_i, D_i, \{Runnable_j\}, ASIL_i \rangle, \quad (6)$$

where $\{Runnable_j\}$ is the set of runnables assigned to the $Task_i$, $WCET_i$ is the worst case execution time of the task, T_i is the period of the task, D_i is its deadline, and $ASIL_i$ captures the ASIL.

With mixed-criticality functions, the safety functions of different ASILs should be separated from each other. Spatial and temporal partitioning must be assured such that a lower criticality task, for example, should not disturb a higher criticality task. Spatial partitioning in the AUTOSAR framework is achieved through the concept of an *OS-Application*. Each OS-Application will use, for example, a different memory area and the AUTOSAR OS will assure that a runnable executing in one OS-Application cannot modify a memory region from another OS-Application. An OS-Application consists of Tasks and has an ASIL:

$$OS\text{-}Application_i = \langle \{Task_j\}, ASIL_i \rangle$$

In order to isolate runnables with different ASIL levels, in our mapping solution, we only allow runnables with the same ASIL level to be assigned to the same Task. Furthermore, the tasks in an OS-Application will contain only tasks with the same ASIL level.

The Tasks are scheduled using a fixed-priority preemptive scheduling scheme, according to AUTOSAR OS specification for multicore ECUs [4]. The communication between runnables that exchange data signals is done through the AUTOSAR RTE (Runtime Environment), see fig. 4 for an example. Runnables are communicating through sender-receiver ports and, at the AUTOSAR Infrastructure layer, it can be observed that the RTE is relying on the Basic Software (BSW) to send data between ECUs.

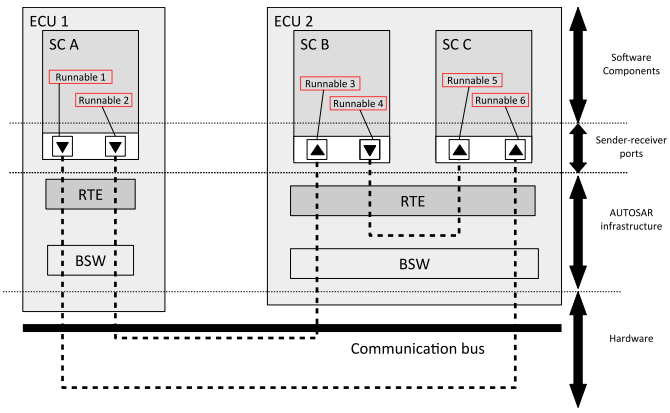


Figure 4: Runnable communication over AUTOSAR Runtime Environment (RTE)

Considering the RTE, we distinguish the following types of communication: (1) Intra-ECU communication, between runnables exchanging data signals mapped on the same ECU, which can be of two types, (1.1) Intra-Core communication, between runnables exchanging data signals mapped on the same core and (1.2) Inter-Core communication, between runnables

mapped on different cores of the same ECU. (2) Inter-ECU communication, between runnables exchanging data signals mapped on different ECUs.

We assume that the runnables use the AUTOSAR sender-receiver with last-is-best mode of communication, i.e., runnables exchanging data signals are using asynchronous communication with non-blocking read/write operations. However, we can model all the AUTOSAR communication scenarios.

Problem Formulation

Given an application model *Application* and an architecture model *Architecture*, as presented in the previous sections, we are interested to determine:

- A mapping M_{ECU} of software components to ECUs,
- A mapping M_{core} of runnables to cores,
- A mapping M_{tasks} of runnables to Tasks,
- A mapping M_{apps} of Tasks to OS-Applications,

such that the following objectives are minimized:

O1 The overall communication *bandwidth*. We prefer solutions where the runnables that communicate are mapped "near" each other, reducing the need for communicating signals over the on-chip and off-chip buses.

O2 The *variance* of the core utilization of the system. If we have an ECU with three computational cores, we want that the runnables are mapped such that the utilization is balanced, e.g., one core does not have an utilization of 80% and the rest of the cores have 10%. We compute the variance of the core utilization to measure how far the values are from the overall mean.

Further, the following constraints should be satisfied:

C1 The mapping constraints are satisfied. The constraints may be imposed by the system integrator or by the AUTOSAR model. For exemple, an AUTOSAR constraint specifies that runnables from the same software component have to be mapped on the same ECU.

C2 The OS-Applications are schedulable. The mapping has to be done such that the Tasks and messages are schedulable.

C3 The runnables with different ASILs are separated, as discussed earlier, i.e., Tasks contain runnables with the same ASIL and OS-Applications may contain only Tasks with the same ASIL.

Considering the application from fig. 3 and tables 1 and 2, to be mapped in the architecture from fig. 2, in fig. 5 we have a possible mapping solution. Thus, *Software Component*$_1$ is mapped to ECU_1 and *Software Component*$_2$ to ECU_2. Next, all runnables from the *Software Component*$_1$ will be mapped to $Core_1$ and a possible mapping for runnables from *Software Component*$_2$ to the cores of the ECU_2 is as follows: *Speed Setpoint*, *Application condition* and *Limp home* mapped to $Core_2$ and *Basic function* and *Controller* mapped to $Core_3$.

After all the runnables are mapped to the cores, they must be

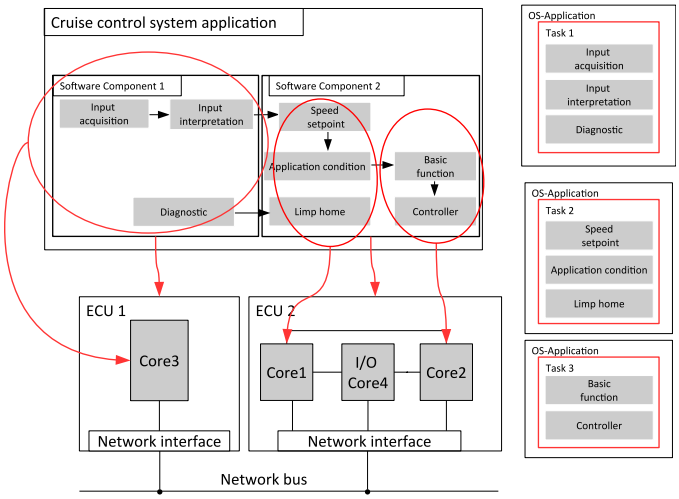


Figure 5: Example mapping solution

grouped into Tasks. Runnables with the same ASIL level, will be grouped into the same task on each processing core, as depicted in the figure. Thus, we have $Task_1$ with the runnables *Input aquisiton*, *Input interpretation* and *Diagnostic* on $Core_1$, $Task_2$ composed of *Speed setpoint*, *Application condition* and *Limp home* on $Core_2$ and $Task_3$ with *Basic function* and *Controller* mapped on $Core_3$. The last step is to group the Tasks into OS-Applications for each core. The Tasks with the same ASIL level will be grouped into the same OS-Application to isolate runnables with different ASIL levels from each other, as depicted in the figure.

Schedulability Analysis

To determine the schedulability of a mapping solution, several approaches are possible. Practitioners are using simulation to evaluate the end-to-end delays in a system, but without providing guarantees. Researchers have proposed end-to-end worst case analysis solutions for AUTOSAR applications [23]. However, in this paper for simplicity, we use the utilization-based schedulability test for independent tasks proposed by Liu & Layland [16]. Thus, a sufficient condition for the tasks to be schedulable on a given core is $U_{core} \leq n \times (2^{\frac{1}{n}} - 1)$, where U_{core} is the core utilization and n is the number of tasks mapped to the core. Note that in our application model, there are data dependencies between runnables. Our assumption is that the runnable that produces the data (sender) does not block waiting for an answer that the data signal has been received by the consumer runnables. Also, a runnable that consumes the data (receiver) is not waiting in a blocked state for the data to arrive, but uses the latest value of the signal. This is the most common type of communication in automotive applications.

The utilization U_i of a $Task_i$ is defined as its WCET over its period, i.e., $U_i = WCET_i/T_i$. Since in the worst-case all the runnables inside a Task will be executed, we have to assume that the WCET of a task is the sum of the WCETs of its runnables. Further, we consider that the period T_i of a $Task_i$ is the greatest common divisor of the runnable periods.

However, instead of computing the processing core utilization

using the task utilizations, we are computing it using the utilizations of the corresponding runnables. The motivation for using the "runnable view" for computing the core utilization is that since not all runnables are activated when the task is executed, the utilization might be over-estimated in case we are using the "task view". As an example, if we have three runnables that are grouped into a task and the runnables have the following properties: $Runnable_1$ has $WCET_1 = 5\ ms$ and $T_1 = 25\ ms$, $Runnable_2$ has $WCET_2 = 1\ ms$ and $T_2 = 50\ ms$ and $Runnable_3$ has $WCET_3 = 10\ ms$ and $T_3 = 100\ ms$, then the utilization of the task is $WCET/T$ where $WCET = WCET_1 + WCET_2 + WCET_3$ and $T = gcd(T_1, T_2, T_3)$. Therefore, the utilization is: $U_{task\ view} = \frac{5+1+10}{gcd(25,50,100)} = \frac{16}{25}$, which is pessimistic, since the runnables will not require such a high computation from the processing core. With the proposed "runnable view" the utilization is instead $U_{runnable\ view} = \frac{WCET_1}{T_1} + \frac{WCET_2}{T_2} + \frac{WCET_3}{T_3} = \frac{5}{25} + \frac{1}{50} + \frac{10}{100} = \frac{8}{25}$.

Thus, the utilization for each core is computed as the sum of all runnable utilizations that are mapped to that core, where m represents the number of runnables assigned to that processing core:

$$U_{core} = \sum_{i=1}^{m} \frac{Runnable_i.WCET}{Runnable_i.T} \tag{7}$$

The related work ignores the time it takes for a runnable to communicate over RTE. However, due to the complexity of the AUTOSAR framework and because of the safety related features introduced for communication, these overheads should not be ignored. A discussion of the overheads introduced by the AUTOSAR RTE is available in [11]. Our model takes into account the particularities of the AUTOSAR RTE, but does not make a distinguish between between signal storage strategies as software variables. Using the SymTA/S timing analysis tool [11], they determined the communication overhead for runnables exchanging signals. The values obtained were taken into account when computing the core utilization. The results show that depending of how runnables exchanging signals are being mapped, the total communication overhead per core can be quite high, e.g., up to 20%.

Depending of how runnables communicate with each other, the RTE layer will use IOC layer (for inter-core communication), COM (communication stack) layer for inter-ECU communication or E2E library for reliable communication. We account for the impact of AUTOSAR on the timing of runnables in the WCET of a Runnable. Thus, we define the WCET of a runnable as composed of $WCET_{computational}$, which is the amount of time needed by a runnable to execute its instructions without interacting with the AUTOSAR RTE, and $WCET_{communication}$, which is the amount of the time spent by a runnable when it is using the RTE layer's API for communication. Based on the type of communication and based on the AUTOSAR *sender-receiver* mode being used, we capture the $WCET_{communication}$ overhead for runnables exchanging data signals as:

- α, if runnables are mapped into the same Task,
- β_0, if runnables have the same ASIL levels and are mapped into different Tasks,
- β_1, if runnables have different ASIL levels and are mapped into different Tasks,
- γ, if the runnables are mapped into Tasks on different cores on the same ECU,
- θ, if the runnables are mapped into Tasks on different ECUs.

The overheads are determined for the largest signal size, i.e., considering the worst-case. We assume that these overheads, which are specific to a given AUTOSAR implementation, have been determined as in [11] and are part of our model. Regarding the schedulaiblity of the communication, we check that the mapping solution does not lead to a bus utlization which is greater than 100%. For each ECU, we define the $Link_{a,b}$ bandwidth for all the links between two cores $Core_a$ and $Core_b$:

$$bandwidth\ Link_{a,b} = \sum \frac{size_{i,k}}{T_i}, \tag{8}$$

where $size_{i,k}$ is the size of a message (signal) from $Runnable_i$ mapped on $Core_a$ to $Runnable_k$ mapped on $Core_b$, and T_i is the period of the sender $Runnable_i$. The *bandwidth Bus* for the inter-ECU bus is defined similarly, considering all the signals exchanged over the bus between ECUs, and taking into account the CAN frame overhead (we assume that each signal is packed in one frame). We also define the utilization of a communication link (or bus) U_{Link} (and U_{Bus}) as the calculated bandwidth over the maximum bandwidth specified in the architecture model.

Simulated Annealing-based Optimization

All of the mapping decisions in our problem (software components to ECUs, runnables to processing cores, runnables to Tasks) can be reduced to a bin-packing problem, which is known to be a combinatorial NP-hard problem [10]. Since our problem is NP-hard, we have decided to use a metaheuristic, since they have been used successfully for solving task mapping problems [6]. Hence, we use a Simulated Annealing-based approach [1] to solve our optimization problem. In this paper our focus is on the problem formulation, hence we have not compared several metaheuristics to determine which one is the most appropriate for the problem; we leave such an investigation for future work. SA is an optimization heuristic that tries to find the global optimum, in terms of the *cost function*, by randomly selecting a new solution from the neighbors of the current solution. The SA algorithm is a variant of the neighborhood search technique, where the local search space is explored by moving from the current solution to a neighbor solution.

We use the *cost function* defined in eq. (9) to guide the Simulated Annealing search. The cost function combines objective **O2**, related to the *core utilization variance* (first term of the equation), with objective **O1**, related to the *communication bandwidth*, using linear scalarization of the two objectives with the weights W_2 and W_1, respectively. The mapping constraints **C1** and separation constraints **C3** are enforced during the design transformations, such that no visited solutions invalidate these constraints. However, regarding the schedulability constraint **C2**, we have decided to allow the exploration of solutions which are not schedulable, in the hope of improving the design space exploration. The schedulability of cores is checked by the third term of eq. (9) and the communication schedulability is checked

by the fourth term. If the application is schedulable, these terms will be zero, and hence ignored by the cost function. However, if the application is not schedulable, these terms are multiplied by large penalty weights, P_1 and P_2, which forces the search to move away from unschedulable solutions.

$$
\begin{aligned}
&cost\ function = W_1 \times \sigma + \\
&W_2 \times \left(\sum_{Link \in NoC_i.Links_i \cup Bus} bandwidth\ Link \right) + \\
&P_1 \times \left(\sum_{core \in Architecture} \max(0, U_{core} - U_{core\ max}) \right) + \\
&P_2 \times \left(\sum_{Link \in NoC_i.Links_i \cup Bus} max(0, U_{Link} - 1) \right)
\end{aligned}
\tag{9}
$$

where

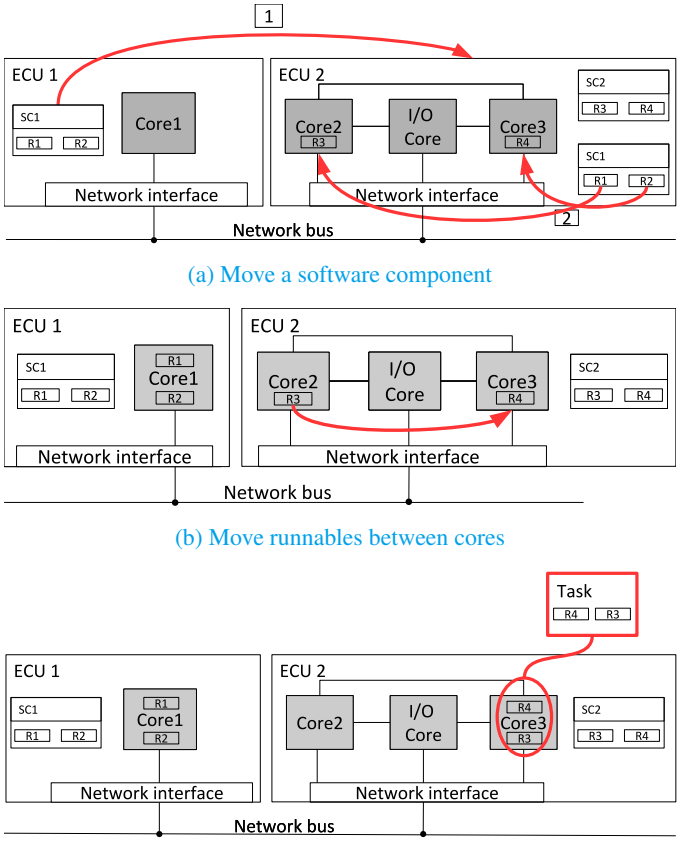
$$
\begin{aligned}
\sigma &= \frac{1}{N-1} \times \left(\sum_{core \in Architecture} (U_{core} - \mu)^2 \right) \\
\mu &= \frac{1}{N} \times \left(\sum_{core \in Architecture} U_{core} \right)
\end{aligned}
\tag{10}
$$

In the cost function, N stands for the total number of cores in the system. The first term computes the variance of the core utilization. The *variance* σ measures how far are the core utilizations from each other. If the variance is zero, it means that all the cores have the same utilization, so the utilization is evenly balanced. A small variance indicates a good load balancing, whereas a large variation indicates that the ultilizations are not well balanced. We want that the mapping of runnables should be done in such a way that each core utilization is closer to the mean utilization μ.

The second term of the equation computes the sum of bandwidth utilization for each inter-ECU and inter-core communication link. We want that the runnables that are exchanging data signals to be mapped as close as possible to each other such that the overall inter-ECU and inter-core communication is minimized. For example, in the case of two runnables exchanging data signals, if they are mapped into the same core, no inter-ECU or inter-core bandwidth will be used.

The third term of the equation adds a penalty factor when one of the core utilizations is higher than a threshold $U_{core\ max}$ given by a system integrator. For the experiments, $U_{core\ max}$ was set to 0.69, which guarantees schedulability according to Liu & Layland bound [16]. Finally, the last term of the equation checks that no communication link has a bandwidth utilization higher than 100%, which would mean that the messages are not schedulable.

An essential component of SA is the generation of a new solution starting from the current one. The neighbor solutions are generated through performing design transformations on the current solution. We have defined three design transformations in fig. 7 that we want to apply to a given solution. At each step, one of the strategies is chosen randomly and applied to



(a) Move a software component

(b) Move runnables between cores

(c) Move runnables into the same Task

Figure 7: Design transformations

the current solution. For our problem, the strategies involve the following, see (a)-(c) in fig. 7 for an illustration:

(a) Randomly choose a software component and map it to a new ECU which is different from the one where it is currently assigned. First, the ECU is selected at random. Second, all the runnables inside the software component are randomly mapped to the cores of the new ECU.
(b) Randomly choose a runnable and map it to a new core. The core is selected at random from the ones of the ECU where the software component that contains the runnable is mapped.
(c) Randomly choose two runnables that reside on the same core and group them together into a Task. At the beginning of the algorithm, all the runnables are mapped into their own Task.

During these transformations we check that the mapping constraints imposed by the system integrator or by the safety requirements, are satisfied. The resulted design transformation is applied to the current solution. In general, the new solution is accepted if it is an improved one. However, in the case of SA, a worse solution can also be accepted with a certain probability that depends on the deterioration of the cost function and on a control parameter called *temperature* which is analog to the temperature concept of the physical annealing process. SA starts from an *Initial Temperature* and stops when it reaches

a *Final Temperature*. The search stays at a *temperature* for a given number of *Steps per Temperature*. After these steps, the temperature is reduced using a *Cooling Factor*.

Experimental Results

For the evaluation of our proposed SA approach (called "Simulated Annealing Mapping", SAM) we have used three realistic case studies, namely a "Cruise control case study" [2], a "PSA (Peugeot-Citroen Automobile Corp.) case study" from [8] and a "Volvo case study" from Volvo Trucks [17]. The details of the case studies are presented in table 3, where the first two columns contain the case study ID and name, and the next three columns list the number of software components, number of runnables and number of signals in each case study. The ASILs vary from ASIL D to QM.

Table 3: Use cases

ID	Name	Software components	Runnables	Signals
CS1	Cruise control	2	8	6
CS2	PSA case study	6	31	17
CS3	Volvo case study	50	75	300

SAM was implemented in C#, running on Windows 8.1 computer with four Intel 64-bit CPUs at 2.4 GHz and 8 GB of RAM. We have used SAM to map the three case studies on the architectures in table 4. The table lists the name of the architecture, the number of ECUs, number of total cores, and the off-chip and on-chip communication bandwidths.

Table 4: Hardware architectures

ID	ECUs	Cores	ECU bandwidth (bytes/s)	Core bandwidth (bytes/s)
Arch1	2	4	50,000	10,000
Arch2	2	6	500,000	100,000
Arch3	1	3	N/A	500,000

For these architectures, we have used the $WCET_{communication}$ overhead values from table 5.

Table 5: $WCET_{communication}$ overheads

Overhead	Value (ms)
α	0.01
β_0	0.02
β_1	0.03
γ	0.04
θ	0.06

The outcome of the simulated annealing heuristic depends on the following parameters: initial and final temperature, the number of iterations per temperature, the cooling factor and the number of steps per temperature. For these parameters we have chosen the values as recommended in [24], see table 6. Also, for the cost function, we have set the weights W_1 and W_2 to 0.5, the maximum core utilization ($U_{core\ max}$) to 0.69 and the penalty factors (P_1 and P_2) to 1,000.

Table 6: Simulated Annealing parameters

Parameter	Value
Initial Temperature	1
Final Temperature	0.00001
Cooling Factor	0.95
Steps per Temperature	100

In the first set of experiments, we were interested to determine the quality of our proposed SAM approach. For this purpose, we have developed an exhaustive search (using backtracking) to obtain the optimal solution. We were able to obtain the optimal solution for small problem sizes such as CS1 mapped on Arch1. Our SAM approach has been able to find this optimal solution in about 0.5 seconds.

In the second set of experiments, we were interested to evaluate the ability of SAM to find solutions in a reasonable time for large case studies. Thus, we have mapped CS1 on Arch1, CS2 on Arch2, and CS3 on Arch3. For all the experiments we have used the SA parameters from table 6, except CS2 where we have increased the steps per temperature to 500. We were able to obtain schedulable solutions for all these cases. The results are summarized in table 7, where in the first two columns we have the application model and the architecture on which it was mapped, respectively, in the third and fourth columns we have the number of tasks and OS-Applications resulted after the mapping, respectively. After the mapping, all the applications were schedulable (as indicated in the fifth column), and all the constraints have been satisfied. The last column shows the runtime of SAM for each of the mapping optimizations. As we can see from the table, our proposed SAM approach is able to find, in a short time, schedulable implementations.

Table 7: Simulated Annealing Mapping (SAM) results

Case study	Arch.	Tasks	OS-Apps.	Sched.	Runtime
CS1	Arch1	7	4	yes	0.5 sec.
CS2	Arch2	19	16	yes	8 sec.
CS3	Arch3	33	3	yes	45 sec.

We have chosen to present an output of the SAM tool for the "PSA (Peugeot-Citroen Automobile Corp.) case study" [8]. The application model has 6 *software components* with a total of 31 *runnables*, see fig. 8 where arrows represent the signals exchanged:

- *Engine Controller*: *runnables*: $F0 - F7$
- *Automatic Gear Box*: *runnables*: $F8 - F11$
- *Anti-locking brake*: *runnables*: $F12 - F17$
- *Wheel angle sensor*: *runnables*: $F18 - F19$
- *Suspension controller*: *runnables*: $F20 - F24$
- *Body work*: *runnables*: $F25 - F31$

Detailed informations about runnable's WCET, period and ASIL level are presented in table 8.

Table 8: Runnable information for automotive application

Runnable	WCET(ms)	Period(ms)	Deadline(ms)	ASIL
F1	2	10	10	C
F2	2	20	20	C
F3	2	100	100	C
F4	2	15	15	C
F5	2	14	14	C
F6	2	50	50	C
F7	2	40	40	C
F8	2	15	15	D
F9	2	15	15	D
F10	2	50	50	D
F11	2	14	14	D
F12	1	20	20	D
F13	2	20	20	D
F14	1	15	15	D
F15	2	100	100	D
F16	1	20	20	D
F17	2	14	14	D
F18	4	14	14	B
F19	4	20	20	B
F20	1	20	20	C
F21	1	20	20	C
F22	1	10	10	C
F23	2	14	14	C
F24	2	15	15	C
F25	2	50	50	A
F26	2	50	50	A
F27	2	10	10	A
F28	2	100	100	A
F29	2	40	40	A
F30	2	20	20	A
F31	2	100	100	A

The architecture model chosen consists of two ECUs with three cores each (fig. 9). The values for the inter-core and inter-ECU communication bandwidth are presented in table 9 and table 10.

Table 9: Inter-core bandwidth

Core Name	Core Name	Bandwidth (bytes/second)
Core1	Core2	100,000
Core1	I/O Core1	100,000
Core2	I/O Core1	100,000
Core3	Core4	100,000
Core3	I/O Core2	100,000
Core4	I/O Core2	100,000

Table 10: Inter-ECU bandwidth

ECU name	ECU name	Bandwidth (bytes/second)
ECU1	ECU2	500,000

The mapping solution of *software components* to ECUs produced by our SAM tool is presented in table 11. The mapping of *runnables* to the ECU's cores is presented in table 12. The number of *runnables* per core is between 4–7 since not all *runnables* could be mapped onto one ECU without having the core utilization constraint broken or the inter-core communication bandwidth exceeded.

Table 11: Software components to ECU mapping

Software component	ECU
Automatic Gear Box ID	ECU1
Suspension controller ID	ECU1
Body work ID	ECU1
Engine Controller	ECU2
Anti-locking brake	ECU2
Wheel angle sensor	ECU2

Table 12: Runnables to core mapping

Runnables	ECU;Core
F10, F21, F23, F24, F25, F28, F29	ECU1;Core1
F11, F22, F27, F31	ECU1;Core2
F8, F9, F20, F26, F30	ECU1;I/O Core1
F10, F21, F23, F24, F25, F28, F29	ECU2;Core3
F3, F4, F6, F7, F12, F13, F17	ECU2;Core4
F5, F14, F15, F18	ECU2;I/O Core2

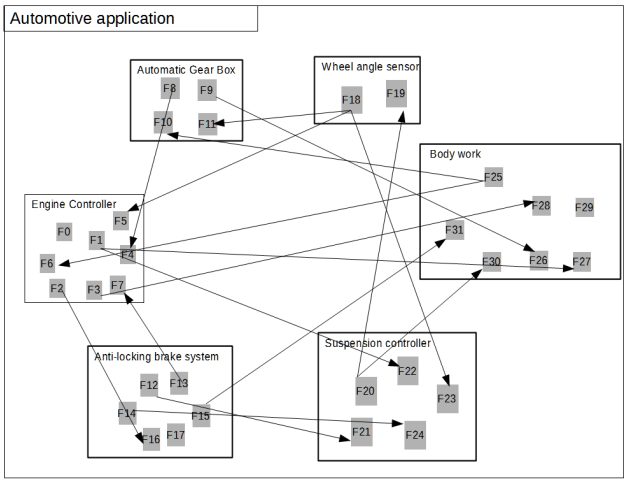


Figure 8: Automotive application

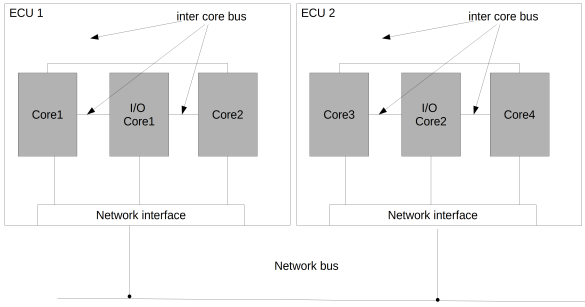


Figure 9: Automotive hardware architecture

Conclusions

In this paper we have presented a Simulated Annealing-based Mapping approach (SAM) for the optimization of mixed-criticality automotive applications on AUTOSAR distributed architectures. The architectures consist of a set of heterogeneous multicore ECUs interconnected by a CAN bus. We consider that the applications are scheduled using fixed-priority preemptive scheduling, and the messages are transmitted according to the CAN protocol.

SAM takes into account the overheads of the AUTOSAR RTE in the WCETs of the runnables, and that the applications of different ASILs have to be separated. SAM determines a mapping such that the utilization of the cores is balanced and the bandwidth requirements are minimized, while at the same time fulfilling the constraints that the applications should be schedulable and the mapping constraints satisfied. Three real life case studies have been used to show the effectiveness of the proposed algorithm.

References

[1] Emile Aarts, Jan Korst, and Wil Michiels. "Simulated Annealing". In: *Search Methodologies*. Ed. by Edmund Burke and Graham Kendall. Springer, 2005, pp. 187–210.

[2] S. Anssi et al. "Enabling Scheduling Analysis for AUTOSAR Systems". In: *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2011 14th IEEE International Symposium on*. 2011, pp. 152–159. DOI: 10.1109/ISORC.2011.28.

[3] AUTOSAR_EXP_VFB. *Virtual Functional Bus*. Tech. rep. AUTOSAR 4.2.1, 2014.

[4] AUTOSAR_SW_OS. *Specification of Operating System*. Tech. rep. AUTOSAR 4.2.1, 2014.

[5] James Barhorst et al. "A Research Agenda for Mixed-Criticality Systems". In: *Cyber-Physical Systems Week*. San Fransisco, CA, 2009.

[6] Tracy D Braun et al. "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems". In: *Journal of Parallel and Distributed Computing* 61.6 (2001), pp. 810 –837.

[7] Alan Burns and Rob Davis. "Mixed Criticality Systems – A Review". In: 2013. URL: http://www-users.cs.york.ac.uk/~burns/review.pdf.

[8] F. Cottet et al. *Scheduling in Real-Time Systems*. ISBN: 9780470847664.

[9] H.R. Faragardi et al. "An efficient scheduling of AUTOSAR runnables to minimize communication cost in multi-core systems". In: *Telecommunications (IST), 2014 7th International Symposium on*. 2014, pp. 41–48. DOI: 10.1109/ISTEL.2014.7000667.

[10] Michael R Garey and David S Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman, San Francisco, CA, USA, 1979.

[11] Peter Gliwa et al. *From Single-Core to Multi-Core Platforms-Systematic Migration of Hard Real-Time Software in AUTOSAR*. 2011, pp. 979–992.

[12] *ISO 11898: Road Vehicles – Controller Area Network (CAN)*. International Organization for Standardization (ISO), Geneva, Switzerland, 2003.

[13] ISO 26262. "ISO 26262 - Road vehicles Functional safety". In: International Organization for Standardization, 2011.

[14] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer, 2011. ISBN: 9781441982360.

[15] Charles E Leiserson and Ilya B Mirman. "How to survive the multicore software revolution (or at least survive the hype)". In: *Cilk Arts, Cambridge* (2008).

[16] C. L. Liu and James W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". In: *J. ACM* 20.1 (Jan. 1973), pp. 46–61. ISSN: 0004-5411.

[17] Florin Maticu. *Functionality assignment to partitioned multi-core architectures*. Tech. rep. Technical University of Denmark, 2015.

[18] N. Navet et al. "Multi-source and multicore automotive ECUs - OS protection mechanisms and scheduling". In: *Industrial Electronics (ISIE), 2010 IEEE International Symposium on*. 2010, pp. 3734–3741. DOI: 10.1109/ISIE.2010.5637677.

[19] Mircea Negrean, Simon Schliecker, and Rolf Ernst. "Timing implications of sharing resources in multicore real-time automotive systems". In: *SAE International Journal of Passenger Cars-Electronic and Electrical Systems* 3.2010-01-0454 (2010), pp. 27–40.

[20] Traian Pop et al. "Timing analysis of the FlexRay communication protocol". In: *Real-Time Systems* 39.1-3 (2008), pp. 205–235.

[21] Salah Eddine Saidi et al. "An ILP Approach for Mapping AUTOSAR Runnables on Multi-core Architectures". In: *Proceedings of the 2015 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*. RAPIDO '15. 2015.

[22] O. Scheickl and M. Rudorfer. "Automotive real time development using a timing-augmented AUTOSAR specification". In: *Proc. ERTS* (2008).

[23] Simon Schliecker et al. "Reliable performance analysis of a multicore multithreaded system-on-chip". In: *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*. ACM. 2008, pp. 161–166.

[24] Steven S. Skiena. *The Algorithm Design Manual*. 2008.

[25] Domiţian Tămaş-Selicean, Paul Pop, and Wilfried Steiner. "Design optimization of TTEthernet-based distributed real-time systems". In: *Real-Time Systems* 51.1 (2015), pp. 1–35.

[26] E. Wozniak et al. "An optimization approach for the synthesis of AUTOSAR architectures". In: *Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference on*. 2013, pp. 1–10. DOI: 10.1109/ETFA.2013.6647952.

Contact Information

Paul Pop, DTU Compute Dept., Technical University of Denmark, paupo@dtu.dk

Acronyms

AUTOSAR	Automotive Open System Architecture
ASIL	Automotive Safety Integrity Level
BSW	AUTOSAR base software that includes the operating system
CAN	Controller Area Network
COM	Comunication Stack
IOC	Inter OS-Application communicator
E2E	End-to-End
ECU	Electronic Control Unit
QM	Quality Managed
RTE	Runtime Environment
SA	Simulated Annealing
SAM	Simulated Annealing Mapping
OS	Operating System
WCET	Worst Case Execution Time